

**BUS252**

**Python for Data Analytics**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**End-of-Course Assessment**

**July 2021 Presentation** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name: Chia Wei Hao, Terry**

**Student ID: H1870852**

**Submission Date: 13 September 2021**

import pandas as pd

import numpy as np

import sqlite3

#part i

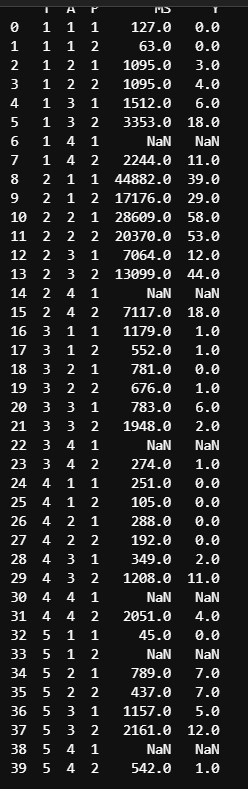
#importing ship.csv into panda data frame and renaming it as shipdf

#declaring. values as NaN (missing values)

ship = pd.read\_csv ('ship.csv', na\_values = ['.'],header = 0)

#print (ship)

Output:

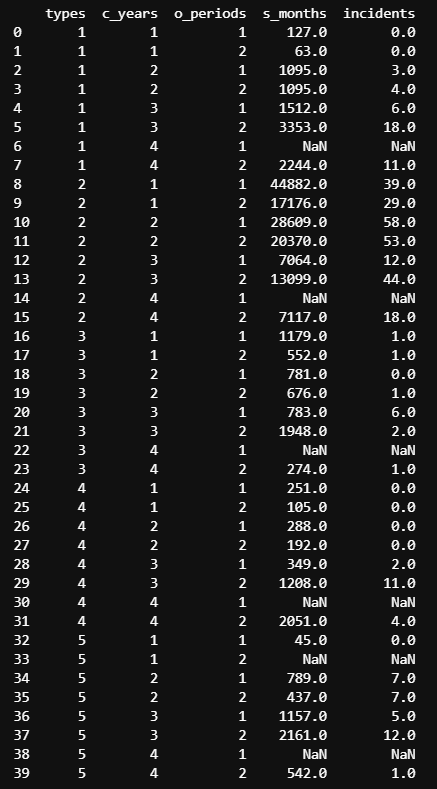


#part ii

ship = ship.rename(columns = {'T':'types', 'A':'c\_years', 'P':'o\_periods', 'MS':'s\_months','Y':'incidents'})

#print (ship)

Output:



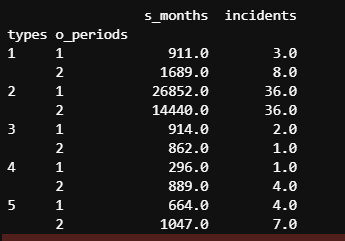
#part iii

#this returns the average for each type and each o\_periods

shipgroup = ship.groupby(['types','o\_periods'])['s\_months','incidents'].mean().round(0)

print (shipgroup)

Output:



#part iv

#identifying the location of the NaNs to be replaced by mean.

inds = np.where(np.isnan(ship))

keyss = np.stack(inds, axis=-1)

print("For each of the rows with Nan values to replace, get the type and o\_periods")

#group the results by types and o\_periods

#use for loop to identify and replace the NaN values with the respective mean from shipgroup

getVal = shipgroup.groupby(['types','o\_periods'])

for name in keyss:

print(name)

#get row with Nan value

element1 = name[0]

element2 = name[1]

print("element1 = ", element1)

row = ship.iloc[[element1]]

# get types to check

rowType = ship.at[element1,'types']

print("rowType = ",rowType)

# get o\_periods to check

rowOperiods = ship.at[element1,'o\_periods']

print("rowOperiods = ",rowOperiods)

print("position in ship replace ",element2)

col = ''

if element2 == 3:

col = 's\_months'

elif element2 == 4:

col = 'incidents'

processval = getVal.get\_group((rowType,rowOperiods))[col]

print("processval",processval)

val = (list(processval))

print("value to put in",val, "at index", element1, "column", element2)

print("updated values")

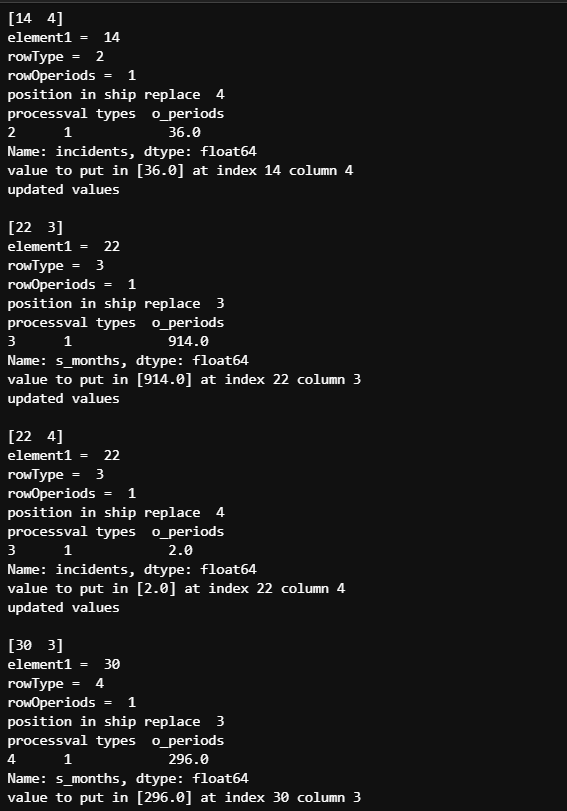
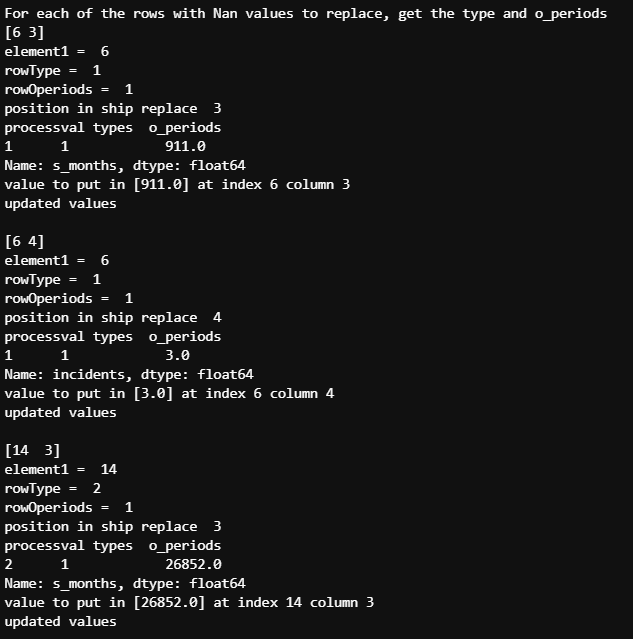
ship.loc[element1,col] = val

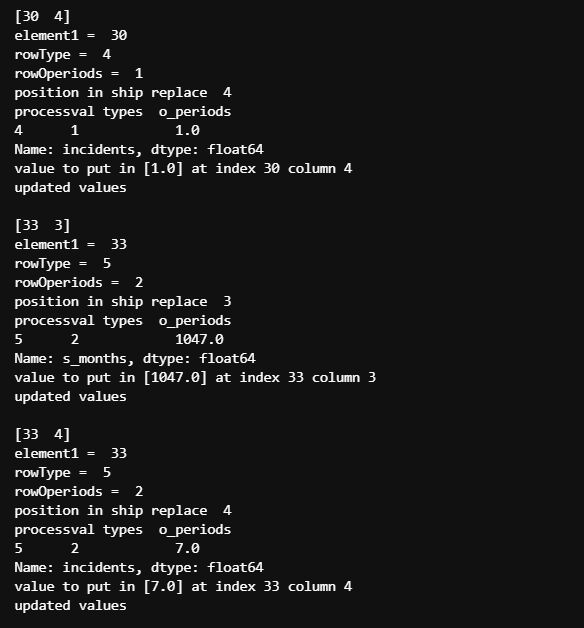
print()

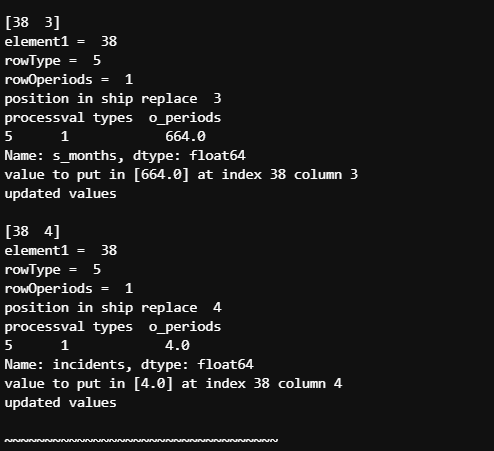
# for incidents

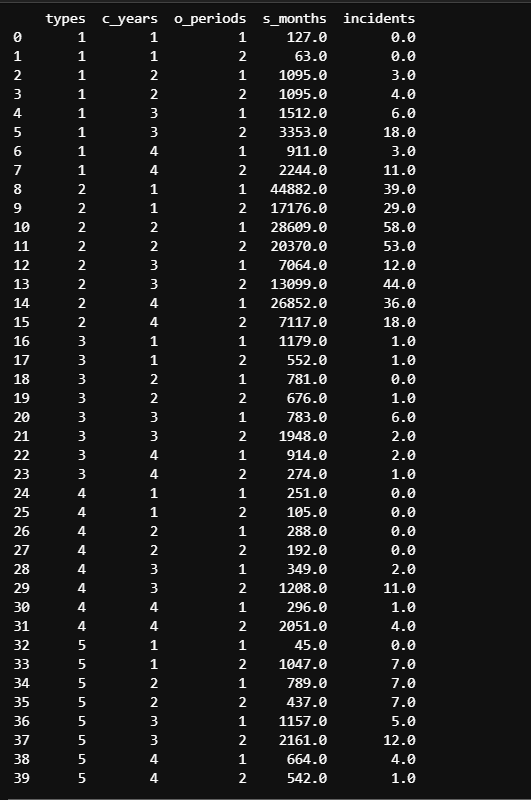
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

print(ship)

Output:  








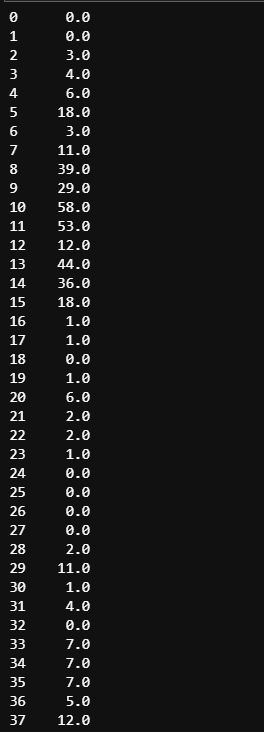
#part v

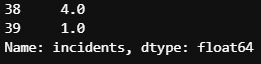
#Y = pd.DataFrame(ship, columns=["incidents"])

Y = ship['incidents']

print (Y)

Output:

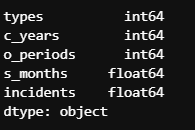




#identifying the current data types

ship.dtypes

Output:



#b part i

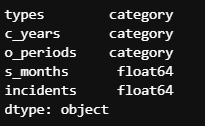
#converting variables to category

convert = ['types', 'c\_years', 'o\_periods'];

ship[convert] = ship[convert].astype('category')

ship.dtypes

Output:



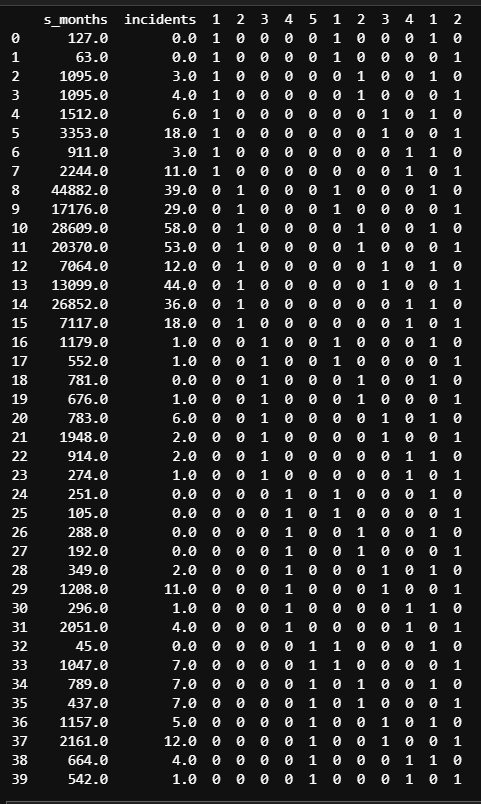
# part ii

#converting categorical data to dummy variables

X = pd.get\_dummies(ship, prefix='', prefix\_sep='',

columns= None)

#print (X)

Output:  


#part iii

#log transform the s\_months to log\_s\_months and insert into both ship and X df.

ship["log\_s\_months"] = np.log(ship["s\_months"])

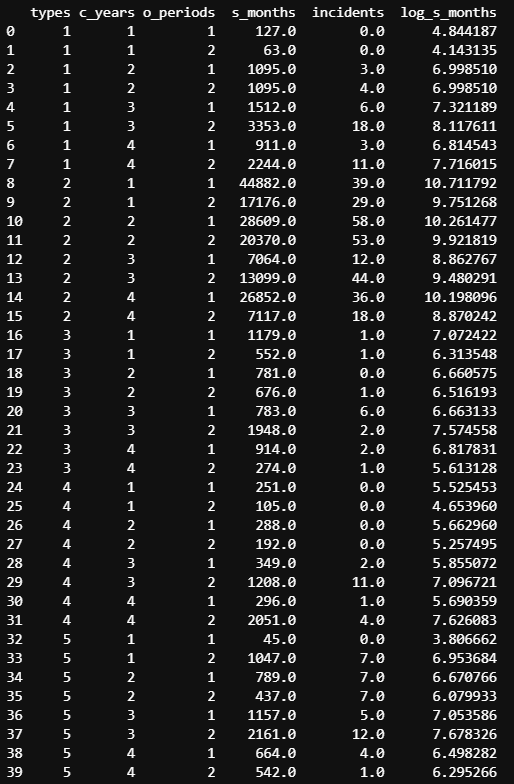
to\_add = ship["log\_s\_months"]

X = pd.concat([X,to\_add], axis = 1)

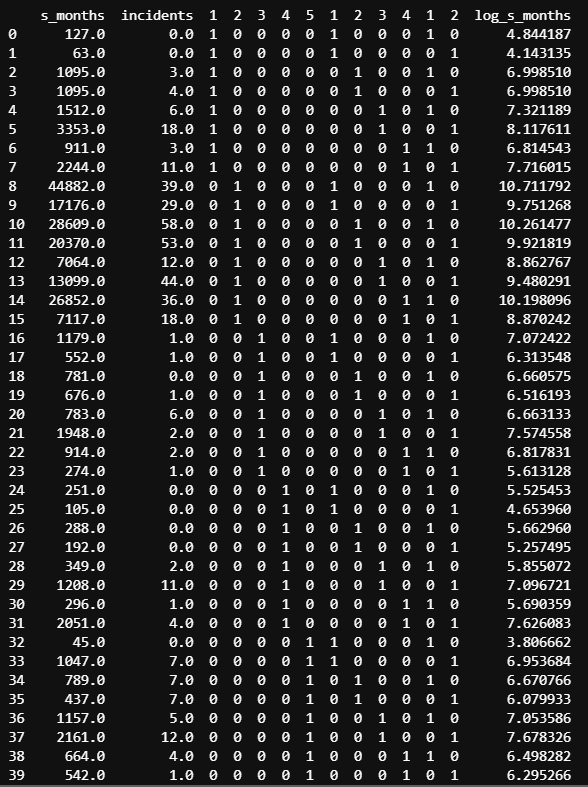
#print (ship)

#print (X)

Output (ship):



Output (X):



#part c

The data set received in the ship.csv is small and it will not have enough data or records for significant results or analysis after splitting the data into training and testing sets, thus, it may not represent the original data set or incidents that have happened.

The results of the modelling could likely either be optimistic (overly good) or pessimisic(overly bad) which will likely return a result that is inaccurate and will not represent the actual precision that is of significance.

#part d

ship.to\_csv("ship\_prepared.csv")

connection\_object = sqlite3.connect("ship")

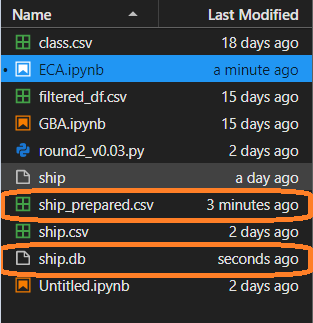
cursor\_object = connection\_object.cursor()

data\_object = pd.read\_csv("ship\_prepared.csv")

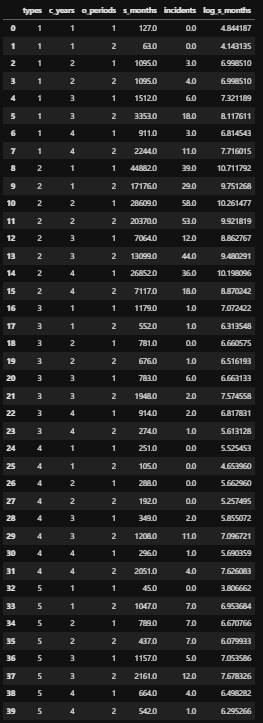
data\_object.to\_sql("ship.db", connection\_object)

ship

Output (relevant files):



Output (table):



#Question 2a

Estimator is a base object that fits a model based on training data which can infer some properties of new data.

Estimator implements the fit method which are either:

the function estimator. fit(x,y) accepts 2 arguments where x = data and y = target

or the function estimator. fit(x) only accepts 1 argument where x = data

The fit() function takes training data as arguments which usually is one array for unsupervised learning or two arrays for supervised learning.

the fit parameter should be resitrcted to data dependent variable.

While the model is fitted using variables X and Y, there is no reference to the variables X and Y.

Additionally, the precomuptered kernels needs to be stored, so that the predict method can be used.

The predict(X) method predicts labels of the data values based on a trained model where X = data.

This function works on top of the trained model and its learned labels to then predict the labels of the data to be tested.

#part b

from sklearn import linear\_model

from sklearn import preprocessing

from sklearn.preprocessing import MinMaxScaler

#converting to a different data type does not work

#convertX = ['log\_s\_months'];

#X[convertX] = X[convertX].astype('int')

#the error returned indicates that the data values are too wide, which is why normalisation is used.

norm = MinMaxScaler().fit(X)

X\_norm = norm.transform(X)

PR = linear\_model.PoissonRegressor(alpha=1.0, fit\_intercept=True, max\_iter=1000, tol=0.001, warm\_start=False, verbose=0)

PR.fit(X\_norm, Y)

\*please see output (poisson regressor)

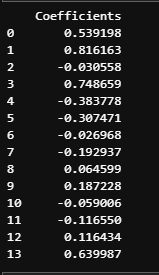
#PR.coef\_ variable to be used.

results = pd.DataFrame(data=PR.coef\_, columns=["Coefficients"])

print (results)

Output (poisson regressor):  


Output:



#part c

#To check answer after results is obtained:

PR.score(X\_norm,Y)

Output:  


#def deviance(X, Y):

# return 2\*metrics.log\_loss(y, linear\_model.predict\_proba(X), normalize=False)

#def inside(y):

# one = y\*np.log\*(y/(math.exp(y)))

# two = y -(math.exp(y))

# final = one - two

# for the sigma portion:

#sum = 0

#for xStep in float\_range(a, User\_x ,alpha):

# cdf\_result = calculate\_pdf(xStep,User\_mean,User\_var)

# sum = sum+cdf\_result

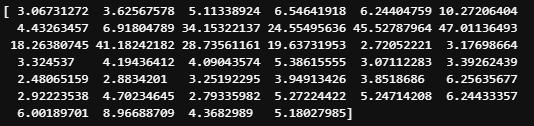
#comments: using def functions does not work as we are missing the required variables for the formula. There is a lack of knowledge for the sequence of the sigma as well.

#predict() is used to check for possible variable results

results1 = PR.predict(X\_norm)

print(results1)

Output:



import statsmodels.api as sm

#poisson\_fit = sm.GLM(Y, X\_norm, family = sm.families.Poisson()).fit()

#poisson\_predict = poisson\_fit.predict()

#print (poisson\_predict)

#result = PR.deviance(X\_norm, Y, var\_weights=1.0, freq\_weights=1.0, scale=1.0)

mod = sm.GLM(Y, X\_norm, family=sm.families.Poisson())

mod.summary

#Comments: GLM method was attempted but was encountered with errors. Additionally, summary() was not successfully processed to provide the required deviance values to be used in the formula processing.

Output:

